

Making a Theme System

Contributed by [dohcan](#)

Update: Due to requests, I have added how you can change your own text, using on input. I hesitated to include it in this tutorial initially, since it requires a slightly different method, and I wanted to stay on topic of using "^" and "haltdef. However, you can see how it is done [here](#).

One of the more popular additions to a script is a theme system. By theme system, I mean, modifying the default output of mIRC for events such as on join, on quit, etc. Adding such a feature to your script will allow it to appeal to a wider audience (and could also be helpful for you personal script, if you get tired of the same old output). This tutorial assumes a pretty good understanding of scripting, and will hopefully show you some new techniques on creating a theme system.

Where Do We Start

The first thing we must do is setup our script to halt the standard mIRC replies. This is done by using a combination of the "^" (caret) event prefix and the haltdef command. A generic example of that might be:

```
on ^*:event: {
    echo -a New Output for event: $1-
    haltdef
}
```

If we look at the above example, the "^" tells mIRC that we *may* want to halt the incoming text and display our own. The haltdef command makes mIRC go ahead and halt the default text. If we had omitted the haltdef command, the original mIRC text for the event would still be shown. Let's look at a real world example:

```
; we want to make everything with the word "joe" in it blue
; otherwise, we want to leave it alone
on ^*:text*:#: {
    if (joe isin $1-) {
        echo 2 $chan < $+ $nick $+ > $1-
        haltdef
    }
    else return
}
```

Now, I know there are several different ways to approach that, but you see how "^" and haltdef are applied. As of mIRC v5.71, the events which can be halted with haltdef and "^" are (from mIRC help file):

ACTION, BAN, CHAT, DEHELP, DEOP, DEVOICE, HELP, INVITE, JOIN, KICK, MODE, NICK, NOTICE, OP, OPEN, PART, PING, TEXT, UNBAN, USERMODE, VOICE, QUIT, SERV, SERVERMODE, SNOTICE, TOPIC, WALLOPS.

It must be noted that even though it isn't in the list, RAWMODE works with "^" and is the preferred method of halting/displaying modes for a channel. However, if you want to show individual modes regardless of how many are set at one time, you can still use the above events that apply to mode changes (op, deop, voice, etc.), and make a trigger for each one.

Part One: Halting the Default Text

Now that we know how to halt the events, let's make our script do that. In this tutorial, we will be using two files, form_halt.mrc and form_disp.mrc, which will be provided at the end of this tutorial,

in their entirety.

form_halt.mrc will contain all of our event hooks, which means they will halt the text and call the procedures to output the custom text. Since there are over 30 different events, I'm not going to cover all of them. I will use JOIN, PART, QUIT, RAWMODE, and TEXT. You will see that most of the events follow the same flow, so after you have learned how to do these, the rest should be fairly easy to finish.

NOTE: All of these events will call their corresponding output procedure, which will be an identifier named \$form.<eventname>, for example, the one for on join will be \$form.join. More about the output procedures will be explained in part two, later in this document.

on join:

```
on ^:join:#: {
    echo $chan $form.join($nick, $address, $chan)
    haltdef
}
```

As you can see, that is quite simple, as will most of these.

However, when handling things like on quit, you must make sure you output to the proper channels, since someone can quit from multiple channels at the same time, but on quit is only called once. To do that, we just use a simple loop and run through the total channels that user is on with us (\$comchan).

```
on ^:quit: {
    var %f.i = 1
    while (%f.i <= $comchan($nick, 0)) {
        %f.chan = $comchan($nick, %f.i)
        echo $comchan($nick, %f.i) $form.quit($nick, $address, $comchan($nick, %f.i), $1-)
        inc %f.i
    }
    haltdef
}
```

Even though we echo'd (possibly) several times, we only need to call haltdef once. You would use almost the exact same method for on nick, but remember to use \$newnick.

Part Two: Displaying the Custom Output

After looking at the above examples, you are probably wondering what the \$form.* means. Well, those are custom identifiers that we have created in our form_disp.mrc. As stated above, for each event that you want to halt, you will create a custom \$form.eventname identifier to go along with it. Then, from within the event, you call that identifier to get your output. So for the above example, the \$form.join may look like this:

```
alias form.join return --> joins[ $+ $3 $+ ] $1 ( $+ $2 $+ ) @ $time(h:nnt)
```

And that will turn

*[22:30] *** IRC-Junkie (junkie@ip.address.host.com) has joined #channel*

into

--> joins[#channel] IRC-Junkie (junkie@ip.address.host.com) @ 10:30p

And of course, you would spruce it up with colors and other control characters to match your style.

Some events have optional messages, such as TOPIC, PART, QUIT, and others. For those, you might make a conditional \$form identifier, such as this one for outputting a new topic:

We assumed it is called with the following syntax

```
$form.topic(nickname, address, channel, [topic text])
```

Where topic text is optional, and a blank topic text means the current topic was unset.

```
alias form.topic {
; if there is topic text
if ($len($4-)) return -!- topic[ $+ $3 $+ ] $1 ( $+ $2 $+ ) changes topic to: $4-
; else, display that the topic was unset
else return -!- topic[ $+ $3 $+ ] $1 ( $+ $2 $+ ) has unset the topic!
}
```

There, we displayed two different outputs, depending on condition of the topic text. You notice I checked the length of \$4- with \$len. This is better than using "if (\$4) ...", because in that case, \$4 could be "0" (a literal zero), after someone had done /topic #channel 0, and the script would evaluate the 0 as false, therefore making it say that the topic was unset, when it really wasn't. Another thing you can do is "if (\$4 != \$null) ...", but I always use "if (\$len(...)) ..." since it is a little shorter.

The beauty of making a separate file for your format identifiers is that you could have several different ones, and make a theme management system, where users could distribute just a single file (form_disp.mrc), and change the entire look of a script. All you would have to do is replace the current form_disp.mrc with the new one and reload it. You could even extend the system to be able to manage several themes simultaneously.

Extra Notes

In this tutorial, we only covered events. There are also raw triggers which are similar, but do not require a "^" prefix and you need to call halt, not haltdef. Same goes for CTCPs. So for example:

```
ctcp *:*:*: {
echo 4 -a CTCP from $nick $+ : $1-
halt
}
```

would halt the standard CTCP display and display that one instead.

Update: Something that is on the same topic, but not done exactly the same way, is altering your own text output. You can do this by hooking into the ON INPUT event. Take this code:

```
on *:input:#: {
if ($left($1, 1) == /) && ($ctrlenter == $false) return
echo -a $form.selftext($active, $1-)
.msg $active $1-
halt
}
```

We first check to make sure the user isn't typing a /command. You want to also check \$ctrlenter, since typing a command with CTRL+Enter pressed makes mIRC say the text instead of using the command. After that, we echo our new \$form.* identifier \$form.selftext, we send a silent message (with .msg), and we halt the event. Halting the event will cause mIRC to not send the text that you sent (since you sent it on your own.)

As said earlier, here is the code for the two example script files we used in this tutorial. Remember, these only cover a few events, but hopefully they will provide a good start.

form_halt.mrc - halts the default text

```
on ^*:text*:#: {
    echo $chan $form.text($nick, $address, $chan, $1-)
    haltdef
}

on ^*:join:#: {
    echo $chan $form.join($nick, $address, $chan)
    haltdef
}

on ^*:part:#: {
    echo $chan $form.part($nick, $address, $chan, $1-)
    haltdef
}

on ^*:quit: {
    var %f.i = 1
    while (%f.i <= $comchan($nick, 0)) {
        %f.chan = $comchan($nick, %f.i)
        echo $comchan($nick, %f.i) $form.quit($nick, $address, $comchan($nick, %f.i), $1-)
        inc %f.i
    }
    haltdef
}

on ^*:mode:#: {
    echo $chan $form.mode($nick, $address, $chan, $1-)
    haltdef
}

; Newly added, ON INPUT
on *:input:~: {
    if ($left($1, 1) == /) && ($ctrlenter == $false) return
    echo -a $form.selftext($active, $1-)
    .msg $active $1-
    halt
}
```

form_disp.mrc - contains the display identifiers for the custom text

```
alias form.join return --> joins[ $+ $3 $+ ] $1 ( $+ $2 $+ ) @ $time(h:nnt)

alias form.part {
    if ($len($4)) return <-- parts[ $+ $3 $+ ] $1 ( $+ $2 $+ ) ( $+ $4- $+ ) @ $time(h:nnt)
    else return <-- parts[ $+ $3 $+ ] $1 ( $+ $2 $+ ) @ $time(h:nnt)
}

alias form.quit {
    if ($len($4)) return <-- quits[ $+ $3 $+ ] $1 ( $+ $2 $+ ) ( $+ $4- $+ ) @ $time(h:nnt)
    else return <-- quits[ $+ $3 $+ ] $1 ( $+ $2 $+ ) @ $time(h:nnt)
}

alias form.mode return -!- mode[ $+ $3 $+ ] $1 ( $+ $2 $+ ) set mode ( $+ $4- $+ ) @ $time(h:nnt)

alias form.text return $time(h:nnt) ( $+ $1 $+ ) $4-

; Newly added, $form.selftext(window, message)
alias form.selftext return $time(h:nnt) ( $+ $me $+ ) $2-
```

All content is copyright by mirccscripts.org and cannot be used without permission. For more details, click [here](#).